Exercises

- Convert the function below to a generic function:

```
function echo(arg) { return arg; }
```

- When compiling the following piece of code, we get an error saying 'Property name does not exist on type T'. How can we solve this problem?

```
function printName<T>(obj: T) {
  console.log(obj.name);
}
```

- An **Entity** should have a unique identifier. The type of identifier, however, is dependent on the use case. In some cases, the ID might be a number, in other cases, it might be a string, GUID, etc. Represent the entity using a generic class.

- Given the following interface, what does **keyof User** return?

```
interface User {
  userId: number;
  username: string;
}
```


Solutions

- Convert the function below to a generic function:

```
function echo<T>(arg: T): T { return arg; }
```

- When compiling the following piece of code, we get an error saying 'Property name does not exist on type T'. How can we solve this problem?

We need to apply a constraint on the generic type parameter so the TypeScript compiler knows that objects of type T have a name property:

```
function printName<T extends { name: string }>(obj: T) {
  console.log(obj.name);
}
```

- An **Entity** should have a unique identifier. The type of identifier, however, is dependent on the use case. In some cases, the ID might be a number, in other cases, it might be a string, and so on. Represent the entity using a generic class.

```
class Entity<T> {
  constructor(public id: T) {}
}
```

- Given the following interface, what does **keyof User** return?

It returns a union of the properties of User: 'userId' | 'username'

